Deep Learning Assignment 13

Ariful Islam

August 2025

Abstract

This report covers tasks of Generative Adversarial Networks (GANs). It covers the generation of synthetic faces using DCGAN, faces with long hair using Conditional GAN, and stylized (painted) faces using CycleGAN. Additionally, it analyzes the mathematical reasoning behind using Binary Cross-Entropy (BCE) loss instead of MiniMax loss in TensorFlow's GAN implementation.

1 Problem Statement

Write a report based on your findings after
generating synthetic faces using Deep Convolutional GAN
generating synthetic faces with long hair using Conditional GAN
generating painted form of faces using CycleGAN
figuring out the reason behind using the BCE loss instead of the MiniMax loss in Tensorflow implementation (mathematical derivation).

2 Introduction

Generative Adversarial Networks (GANs) are powerful tools for creating realistic synthetic images, especially in tasks like face generation. They work by using two parts — a generator that creates images and a discriminator that tries to tell if they're real or fake. In this report, we explore three types of GANs: DCGAN for generating general faces, CGAN for creating faces with long hair by using labels, and CycleGAN for turning photos into painted-style images without needing matched pairs. We also explain why Binary Cross-Entropy (BCE) loss is often used instead of MiniMax loss in TensorFlow, using simple math to support the idea.

3 Datasets

The experiments utilized the following datasets:

- **CelebA** [2]: A large-scale dataset containing over 200,000 celebrity face images with 40 attribute annotations. Used for DCGAN and CGAN tasks to generate synthetic faces and faces with long hair, respectively.
- **Custom Art Dataset**: A collection of painted portraits sourced from public-domain art repositories, used for CycleGAN to learn the transformation from real faces to painted styles.

All datasets were preprocessed to a resolution of 64x64 pixels for DCGAN and CGAN, and 128x128 pixels for CycleGAN, ensuring compatibility with the respective model architectures.

4 Methodology

4.1 Deep Convolutional GAN (DCGAN)

DCGAN is a type of GAN that uses deep convolutional neural networks to generate realistic images. It improves the training stability of regular GANs by replacing fully connected layers with convolutional layers, which are better suited for image data. In DCGAN, the generator learns to create face-like images from random noise, while the discriminator learns to tell if the image is real or fake. Over time, the generator gets better at fooling the discriminator, resulting in realistic synthetic faces.

4.2 Conditional GAN (CGAN)

Conditional GANs take GANs a step further by adding labels or conditions to both the generator and discriminator. This allows the model to generate images based on specific attributes — for example, generating faces with long hair. Instead of just random noise, the generator receives extra information (like the label "long hair"), so it learns to produce images that match that condition. This makes CGANs useful when we want control over the features in the generated images.

4.3 CycleGAN

CycleGAN is used for image-to-image translation without needing paired training data. For example, it can learn to turn regular face photos into painted versions without having exact photo-painting pairs. It uses two generators and two discriminators to convert images from one domain to another and back again, making sure the original content is preserved. This "cycle consistency" helps the model learn meaningful transformations, even with unpaired data.

5 Generating synthetic faces using Deep Convolutional GAN

We used the CelebA dataset containing celebrity face images, resized to 64×64 resolution. The generator and discriminator were trained using a standard GAN structure with the images normalized to [-1, 1]. We trained the model over 120 epochs using a noise vector of size 100, applying Binary Cross-Entropy loss for both generator and discriminator. The training process was tracked by generating and saving sample images at each epoch.

Hyperparameter	Value
Dataset	CelebA
Image Size	64×64
Max Images Loaded	3000
Batch Size	256
Buffer Size	60000
Epochs	120
Noise Dimension	100
Optimizer	Adam (likely)
Number of Samples Saved	16
Activation Scaling	$[-1, 1]$

Table 1: Hyperparameters used in the GAN model training.

Figure 1: In 30 epochs

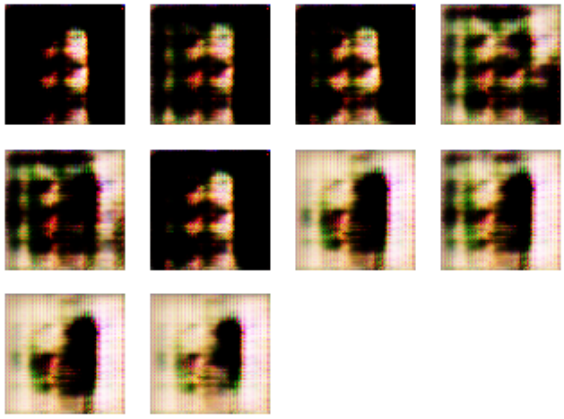

Figure 2: In 60 epochs

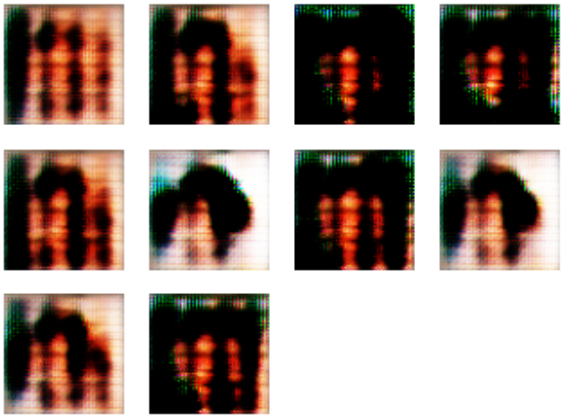

Figure 3: In 90 epochs

Figure 4: In 120 epochs

Figure 5: Generated samples during training

Over 120 epochs, the DCGAN showed clear progress in generating synthetic faces. At first, the images were noisy and lacked structure, but by the end, the faces became more detailed and realistic. The generator and discriminator losses fluctuated significantly, indicating that training was unstable at times. Initially, the discriminator was too strong, which hindered the generator's

ability to learn. Although the final results were much better, the faces still weren't perfect. With more training or better tuning, the model could improve even more.

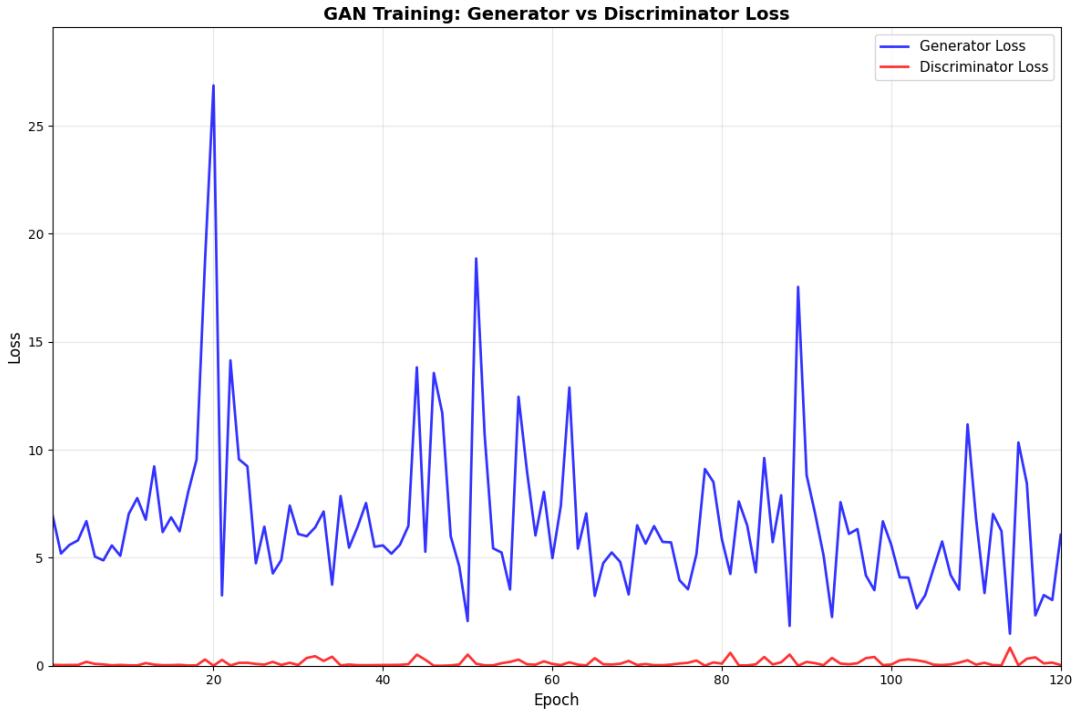

Figure 6: Training loss of Generator and Discriminator

6 Generating synthetic faces with long hair using Conditional GAN

In this task, we used a Conditional GAN to generate synthetic face images based on hair length, distinguishing between short and long hair. We preprocessed the CelebA dataset by resizing images to 64×64 and creating labels from hair-related attributes. The generator takes random noise and hair length labels to produce matching face images, while the discriminator learns to tell real and fake images apart using the same labels. Both models are trained adversarially, improving each other over time. We also saved checkpoints and generated sample images regularly to track progress.

Hyperparameter	Value
Image Size	64×64
Image Channels	3
Latent Dimension	128
Number of Classes	2
Embedding Dimension	64
Batch Size	64
Epochs	350
Learning Rate (Gen)	0.0002
Learning Rate (Disc)	0.0002
Label Smoothing	0.1

Table 2: Hyperparameters used in the Conditional GAN training.

Figure 7: 1 batch smaple image from datasets

The loss curves for the Conditional GAN training, as depicted in the three plots, illustrate the evolving dynamics between the generator and discriminator over 350 epochs. The left plot shows the generator loss starting relatively low but trending upward with fluctuations, suggesting increasing difficulty in fooling the discriminator as it improves. The middle plot indicates the discriminator loss beginning high and steadily declining, reflecting its growing ability to distinguish real from fake images effectively. The right plot, comparing both losses, highlights an imbalance with the discriminator loss decreasing while the generator loss rises, indicating a "tug-of-war" where the discriminator currently dominates, potentially necessitating hyperparameter adjustments to achieve a better balance and enhance the quality of synthetic

images generated.

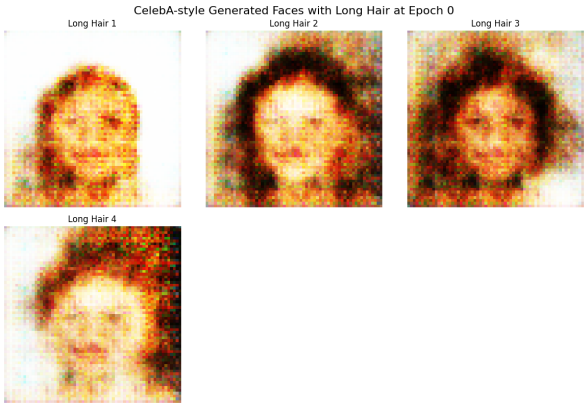

Figure 8: In 50 epochs

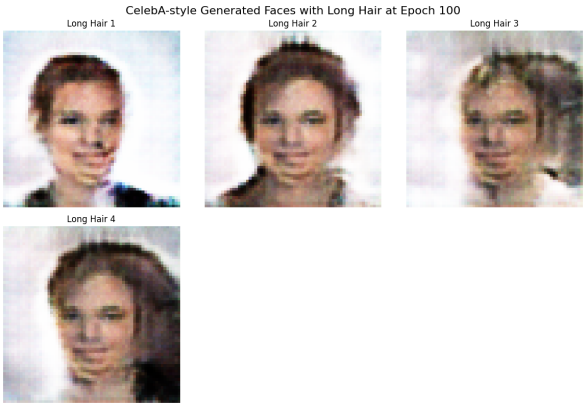

Figure 9: In 100 epochs

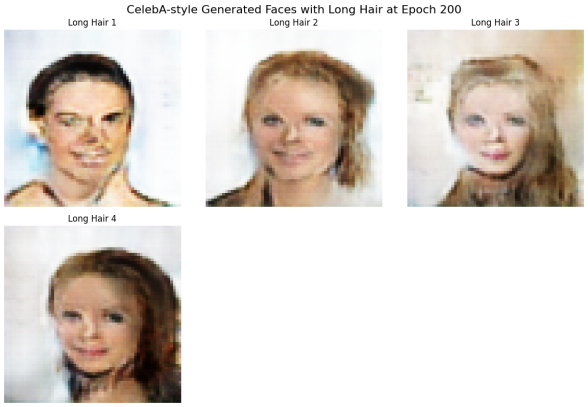

Figure 10: In 200 epochs

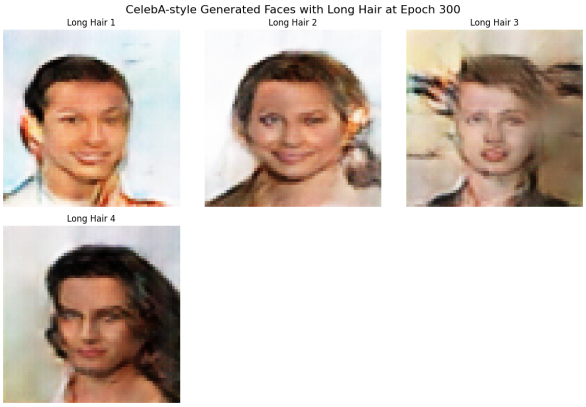

Figure 11: In 300 epochs

Figure 12: Generated long hair samples during training

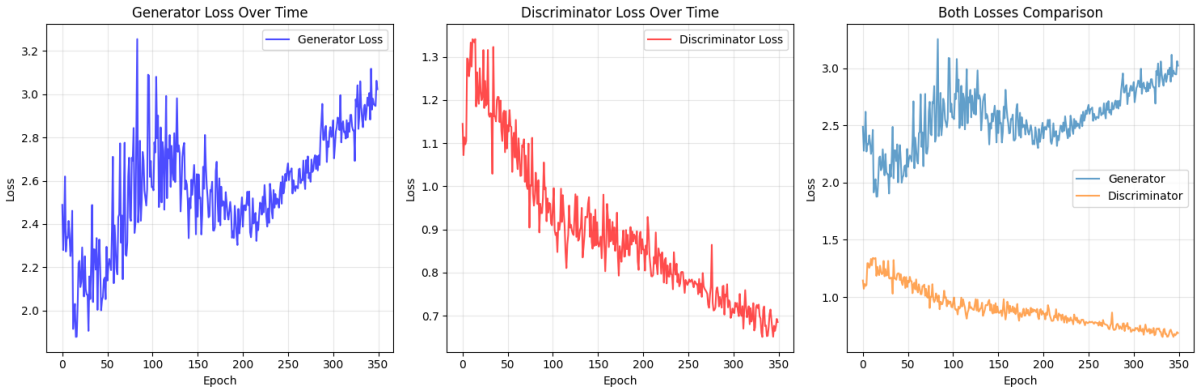

Figure 13: Caption

7 Generating painted form of faces using CycleGAN

In this task, we tackled the task of generating synthetic face images with long hair using a Conditional Generative Adversarial Network (CGAN). We used the CelebA dataset, where images were resized and normalized, and we extracted binary hair-length labels from the attribute annotations (short hair = 0, long hair = 1). Our CGAN model was conditioned on these labels, enabling the generator to create faces with a specific hair type. During training, the generator and discriminator competed: the generator aimed to create realistic, label-conditioned faces, while the discriminator tried to detect fakes. We monitored generator and discriminator loss trends and saved outputs periodically to visually track progress.

Hyperparameter	**Value**
Image Height	64
Image Width	64
Image Channels	3
Batch Size	32
Epochs	40
Learning Rate	2e-4
Beta_1 (Adam)	0.5
Lambda Cycle	10.0
Lambda Identity	0.5

Table 3: Configuration hyperparameters for CycleGAN training.

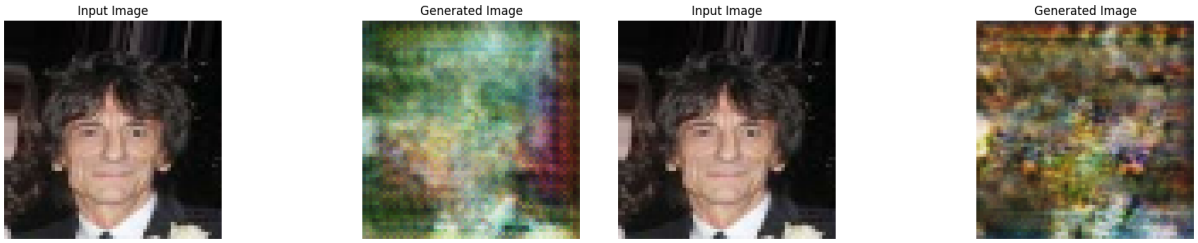

Figure 14: In 50 epochs

Figure 15: In 100 epochs

Figure 16: In 200 epochs

Figure 17: In 300 epochs

Figure 18: Generated painted image samples during training

During the 40 epochs of training our CycleGAN model, we observed a consistent improvement in both generator and discriminator performance. The generator losses for the mappings $G : X \to Y$ and $F : Y \to X$ started at 5.36 and 5.51, respectively, gradually stabilizing around 5.9 and 5.5 by the end of training. This suggests that the generators were learning meaningful transformations between the two domains. Meanwhile, the discriminator losses (D_X and

D_Y) decreased significantly—from approximately 0.55–0.60 at the start to around 0.39–0.44 in the final epochs—indicating better discrimination of real versus generated images. Overall, the convergence of generator and discriminator losses over time reflects balanced adversarial training and supports the generation of high-quality domain translations.

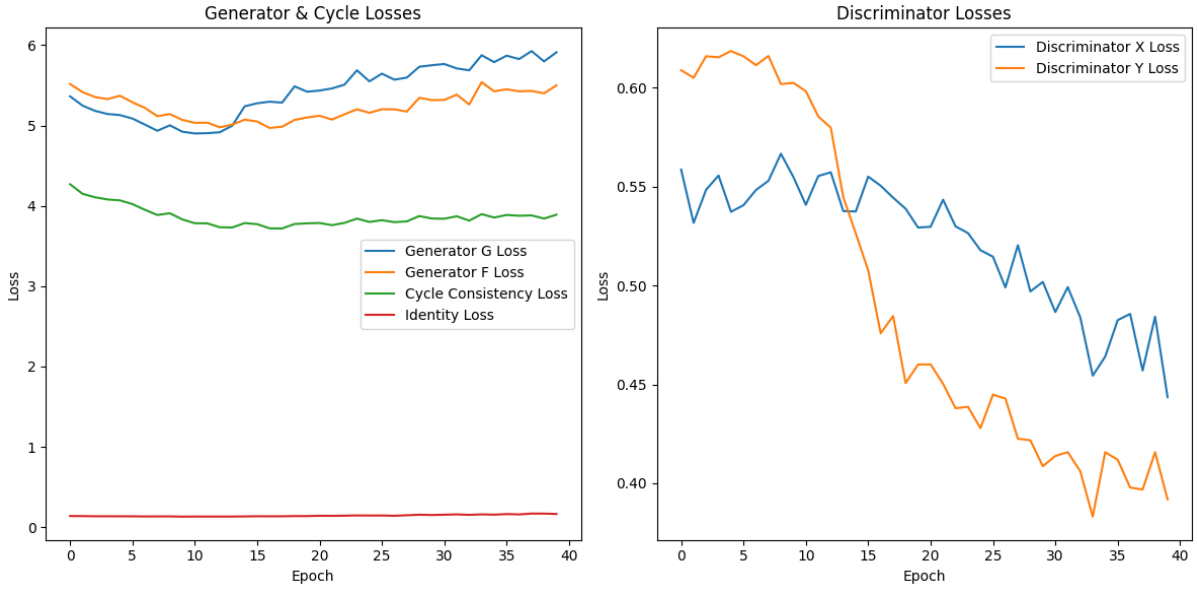

Figure 19: losses of Generator and Discriminator

7.1 BCE Loss vs. Minimax Loss

To understand the preference for BCE loss in TensorFlow implementations, we derive the loss functions mathematically. The original Minimax loss for GANs, as proposed by **(author?)** [1], is defined as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

This loss encourages the discriminator to maximize the probability of correctly classifying real and fake samples, while the generator minimizes the probability of the discriminator detecting fake samples.

However, the Minimax loss can lead to vanishing gradients for the generator when $D(G(z)) \approx 0$. To address this, the non-saturating loss (equivalent to BCE loss for the generator) is used:

$$L_G = -\mathbb{E}_{z\sim p_z(z)}[\log D(G(z))]$$

For the discriminator, the BCE loss is:

$$L_D = -\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

The BCE loss stabilizes training by providing stronger gradients, especially early in training when the generator's outputs are poor. TensorFlow's implementation favors BCE loss for its numerical stability and compatibility with optimization libraries.

8 Conclusion

In this report, we try to explain how we did the 4 task that are assign and how the results and models behave the the parameters. We try to analyze the behavior of each model during training and discussed the results achieved, highlighting how the models improved over time and responded to the chosen hyperparameters.

9 Source Code link

Deep Learning AI GitHub Repository.

10 Reference for writing equation and tables generation

[1] ChatGPT
Overleaf

References

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 27.

[2] Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep Learning Face Attributes in the Wild. *Proceedings of the IEEE International Conference on Computer Vision*, 3730–3738.

[3] Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4401–4410.